Practical throughput estimation for parallel databases

Practical Throughput Estimation for Parallel Databases

by Shaoyu Zhou, M. Howard Williams and Hamish Taylor

Methods for estimating the performance of DBMSs can aid the design of database systems by identifying potential performance bottlenecks or by predicting the relative performance of different designs. Performance estimation is critical in parallel database systems with distributed memory where an effective overall performance depends on a good choice among a wide range of ways of placing data. This paper describes an approach to performance estimation for shared-nothing parallel database systems. It estimates system throughput for a given benchmark or set of queries, and can exercise different data placement schemes to determine the data layout which provides the best throughput value.

1 Introduction

Performance estimation is a valuable technique for the design and tuning of parallel database systems. It allows alternative design choices to be tested easily and cheaply and can highlight potential performance problems in designs long before any construction costs are incurred. A number of approaches have been developed to estimate performance using measures such as throughput or response time to handle a particular workload on some parallel database architectures. An example is APE (Adaptive Performance Evaluator) [1], which was designed to serve as the cost evaluation component for the query optimiser on DBS3 [2]. It includes four parts: KERNEL (a multi-environment cost evaluator of query execution plans), PROF (a statistical profiler of applications), SPEC (a navigational user interface for environment specification) and PERF (a graphical interface). APE uses an analytical model to estimate the response time for query execution. More accurate modelling of query execution can be achieved using simulation methods [3] but only at the cost of time-consuming simulation runs which do not directly predict average or typical behaviour.

Most analytical performance estimation tools use elapsed time as the cost measure in the estimation. Some of them give an estimation of response time based on the cumulative cost. However, for benchmarks such as TPC-B [4] and TPC-C [5] the key indicator of the performance of a database system is the system throughput, defined as the number of transactions completed in a second. A critical part in estimating throughput analytically is the identification of bottlenecks in the system which can be used to calculate the maximum number of transactions whose resource requests on the bottlenecks can be handled within a second. This requires an estimation of the consumption of each resource in a transaction.

Where benchmarks such as TPC-B and TPC-C are to be used at the design and prototyping stage of a parallel database system, an estimation of throughput for these benchmarks is required. An analytical tool can considerably reduce the effort required to accomplish this. Furthermore, system throughput is also an important factor to consider in On Line Transaction Processing (OLTP) applications. Users may want to know the throughput values for their applications on different DBMS architectures. An analytical tool will certainly help them to perform such evaluations. Finally, users may wish to find a relatively good data layout in a shared-nothing parallel system [6] before actually loading the data onto the machine. An analytical

tool is able to predict system throughput values, given different data placement strategies, so that users can choose the best data placement for their applications.

This paper describes an analytical system throughput estimator, which is called STEADY (System Throughput Estimator for Advanced Database sYstems). It can estimate system throughput for a given benchmark or set of queries over a given database under a given DBMS architecture and a given data placement strategy. The result is expressed in transactions per second (tps). It is designed to allow users of parallel database systems to exercise different data placement strategies in order to obtain the best placement scheme which maximises throughput. It is designed as a tool for use by both DBMS designers and database administrators.

This paper is organised as follows. Section 2 introduces some concepts of performance analysis for parallel database systems and raises some issues which need to be investigated in the area of analytical performance evaluation of parallel database systems. Section 3 describes the general architecture of the STEADY system, which provides a flexible environment for performance analysis for parallel database systems. The four major components of STEADY - DPtool, the Profiler, the Modeller and the Evaluator are described in this section. Section 4 lists some areas in which STEADY may be used. The use of STEADY in an effectiveness study of data placement strategies is described in Section 5.

2 Analytical Performance Estimation of Parallel Database Systems

Performance estimation is used for a number of purposes at various stages during the development and delivery of a DBMS product. These include:

1. **Product development**. During the course of development of a large DBMS, various related components such as the DBMS kernel and the supporting operating system are gradually developed towards their target capabilities. The performance of the DBMS needs to be measured at each stage to provide feedback to designers. However, at early stages, unoptimised, incomplete or missing system components substantially affect the performance. An analytical performance estimation tool can be used to assess these performance measures for comparison against their expected values. It can also indicate the system bottlenecks at each stage so that effort can be targetted at alleviating them.

2. **Product validation**. The aim of product validation is to prove product quality before delivery. A series of tests is run on the DBMS to determine whether the product meets its performance requirements. Again, an analytical performance estimation tool provides a convenient means to obtain the expected target performance values for these tests.

3. **System management**. Usually, a database administrator uses historical data on a system's performance to observe trends which may lead to changes in data layout, and bases extrapolations on historical data to estimate the effect of change. Real-time data is then used to manage workload and tune the system after the changes are made, allowing hot-spots and problems to be identified and corrected. It is clearly advantageous to use an analytical tool for this purpose before actually re-organising the live system.

Database performance benchmarks provide an important measure for comparing the performance of DBMSs. They can be used during the development of a DBMS to analyse its performance characteristics. A flexible analytical performance estimation tool should be able to handle multiple benchmarks. In addition to this, when the tool is used for system management, it may be used to estimate the system performance for new applications. Hence, the tool should also be capable of taking new applications as input.

The performance obtained from a parallel database system depends not only on the characteristics of

the DBMS architecture and the schemas of users' applications, but also on the way in which users' data is distributed among the processors in the system. Poor schemes of data placement can seriously affect performance by producing unnecessary bottlenecks in the system. The effectiveness of a particular data placement strategy may also change with time. Since laying out the data of a parallel system is a costly task which every administrator must do, it is desirable that the administrator can rapidly experiment with different data placement schemes on an analytical performance estimation tool to assess which data placement scheme is most effective.

To take account of data placement in performance estimation for parallel databases, it is necessary for the analytical tool to be capable of quantifying the benefit of different data distributions for a selection of workloads. This requires estimation of the consumption of each individual resource such as disc usage and network traffic in order to detect uneven distribution of work across the system. This is also an essential requirement for the tool to produce estimates of system throughput. Therefore, an analytical performance estimation tool for parallel databases should be able to cope with individual resource consumption analysis.

2.1 Database performance benchmarks

Database performance benchmarks can usually be classified into two types: update-intensive transaction processing and the ad hoc query processing of decision support [7]. Transaction processing benchmarks, such as TPC-B, are designed to test OLTP (On Line Transaction Processing) operations, which refer to any kind of applications whose database operations are pre-defined and grouped in transactions. Decision support benchmarks, such as AS3AP [8], are designed to test ad hoc querying on a carefully constructed database. The semantics of the enterprises are implicit in the database and the benchmark programs.

System throughput is a critical performance indicator for transaction processing benchmarks such as TPC-B and TPC-C. Both benchmarks exercise the database components necessary to perform tasks associated with transaction processing environments emphasizing update-intensive database services.

The application in TPC-B is a hypothetical bank, which has one or more branches. Each branch has multiple tellers. The bank has many customers, each with an account. The database represents the cash position of each entity (branch, teller, and account) and a history of recent transactions run by the bank. The transaction represents the work done when a customer makes a deposit or withdrawal against his account. The transaction is performed by a teller at some branch.

The TPC-C benchmark is a more complicated benchmark. It is a mixture of read-only and update-intensive transactions that simulate the activities found in complex OLTP application environments. TPC-C is designed to exercise system functionalities of OLTP applications in a life-like context. The workload is centred around the activity of a wholesale supplier, and includes operations such as verifying and updating customer credit, receiving customer payments, entering and delivering orders, checking on order status, and controlling the inventory of multiple warehouses.

2.2 Data placement

Various data placement strategies have been developed by researchers to try to realise much of the performance potential of shared-nothing database systems. Since the complexity of the data placement problem is NP complete [9], there is no guarantee that an optimal solution can be found in a reasonable amount of time. Therefore, researchers have been working on various heuristic methods to achieve the objectives of data placement.

Generally, a *static* data placement strategy can be divided into three phases. They are:

1. **Declustering** phase in which relations are partitioned into a number of fragments which will be allocated to the discs of a parallel machine in the later phases;
2. **Placement** phase in which the fragments obtained from the declustering phase are allocated to the discs of a parallel machine;

3. **Re-distribution** phase in which data is redistributed to restore good system performance after insertions and deletions made to the local data in each disc have unbalanced the load and degraded the overall system performance.

Dynamic data placement is concerned with dynamically re-organising relations during execution in order to increase the degree of parallelism, reduce response time and improve throughput. It usually takes the form of generating temporary relations (i.e. intermediate relations). The placement of base relations resulting from the initial data placement scheme is not changed.

Quite a variety of data placement strategies have either been adopted by shared nothing parallel database systems such as Bubba [10], Gamma [11] and EDBS [12] or proposed in the literature [13, 14, 15, 16, 17, 18]. None of these data placement strategies is better than all others in every circumstance. Different profiles of data, database queries, database management systems and platform configurations suit the use of different placement strategies. There is a clear need for a tool to try out different strategies and their variations and to compare their relative effectiveness in the particular context in a quick and cheap fashion. Their differences are primarily determined by the ways they go about declustering and placing relation fragments.

Declustering schemes usually partition relations into sets of tuples that either share ranges of values of significant attributes or are grouped by the results of hashing on such attributes. Range partitioning suits performing selections in parallel on the values of attributes that partition the relation over several processors [11, 18]. Hash partitioning suits executing hash joins [19] in parallel on the partitioning attribute where the hash buckets are colocated on the processors involved [16].

Generally placement schemes either aim to minimise an aspect of the load such as the communications costs of moving data around [13], or aim to balance the processing load across the system using a greedy algorithm [14, 15, 17]. Load balancing schemes try to place relation fragments across the processors either to equalise the amount of data stored at each node [14, 15] or to equalise the cumulative frequency of access to data fragments at each node or a combination of both [17].

Most placement schemes can be used in a redistribution phase. However, the cost of redistributing data will often be higher than the extra cost of processing data under skewed distributions. While data redistribution can be done in the background on lightly loaded systems, throughput is likely to be severely degraded if the system is heavily loaded. Some redistribution schemes aim to make redistribution infrequent by calculating the relative benefits of redistributing and only doing so if there is a net advantage [17]. Other schemes aim to minimise the amount of data moved [15].

2.3 Workload analysis

One approach to estimating individual resource consumption for a parallel database is to produce a detailed breakdown of various costs related to system components in a workload and then map these costs into loads associated with the individual system components to give the consumption of each resource in the system. An example of such a breakdown of costs is a workload profile which consists of a number of resource demands from a transaction, such as usage time required on each disc, processing unit, communication control unit and the number of bytes to be transmitted across the network. To produce these costs, a detailed analysis of each query in the benchmark or application is required. Based on the data operations in the query, placement of relations and DBMS execution policy, an estimation of these costs can be produced.

Query analysis is based on a query processing technique called query decomposition. Each data operation in the query is analysed and the costs for all data operations are accumulated to produce the total cost for the query. An important issue related to the analysis of each individual data operation is to estimate how many tuples it produces as output, which might be used as input to following operations.

Statistical profiling is used to assist in estimating

the size of the results produced by a data operation. A base relation profile is built by surveying the characteristics of a relation stored in the database, while a temporary relation profile has to be estimated. A typical profile may contain relation information such as the number of tuples, the number of attributes and keys, attribute information such as the number of distinct values, minimum value, maximum value, value distribution, and index information such as the indexing strategy and key size.

The statistical method used for determining distributions of values during profiling can be classified into two categories [20]: *common parametric distributions* and *nonparametric estimation*. The former utilises some standard statistical models such as the uniform, normal, Pearson family, and Zipf distributions. The estimation of temporary relation profiles can be based on the value distribution type of relation attributes involved in data operations. The most common nonparametric estimation method is the histogram, which divides the range of values of an attribute into intervals. It then tabulates the number of tuples falling into each interval. The numbers of tuple and interval boundaries can be stored as a distribution table.

3 A Flexible Analytical Throughput Estimator

In order to estimate throughput for a given set of queries applied to a given database operating under a given DBMS architecture and a given data placement strategy, an analytical performance estimation tool, STEADY (System Throughput Estimator for Advanced Database sYstems), has been developed. It consists of four components: *DPtool*, *Profiler*, *Modeller* and *Evaluator*.

1. DPtool is a data placement tool which produces data placement schemes for shared-nothing parallel databases. Once a user has selected a strategy, DPtool takes information about the relations and the operations to be performed on them and decides how the relations should be fragmented and divided between the different processing elements. DPtool supports a number of data placement strategies such as Hash, Round-Robin, Hua [15] and Bubba [17].

2. The Profiler is a statistical tool which maintains both base and temporary relation profiles. Its main role is to estimate the number of result tuples produced by a data operation on the database relations (both base and temporary).

3. The Modeller takes as input a benchmark (or set of queries) and analyses them against the data placement scheme and profile information to produce estimates of the various costs for an average transaction. These include I/O costs and data operation costs. I/O costs are in the form of numbers of page reads/writes and of lock requests/replies. The Modeller also estimates the numbers of remote and local I/O operations based on the placement of base relations. These costs are input to the Evaluator.

4. The Evaluator maps the costs into the consumption of each individual resource in the system and then identifies the bottlenecks in the whole system. The Evaluator also calculates overall system throughput values based on estimating the usage of that resource which is a bottleneck to performance in an average transaction.

Fig. 1 illustrates the architecture of STEADY. A typical user session may start with a user having a benchmark which is not contained in the user's benchmark workload profile directory. First DPtool is used to generate a data placement scheme for the benchmark and then the Profiler is executed to generate the base relation profiles. After that, each query in the benchmark is analysed by the Modeller to produce the corresponding benchmark workload profile which can be added to the library of profiles. The analysis may require the assistance of the Profiler for generating temporary relation profiles. Finally, a user may run the Evaluator to identify the resource bottlenecks and to estimate the system throughput for the benchmark.

Using STEADY, a user can obtain system throughput values for a given benchmark under different

data placement schemes. A user can also obtain information on the utilisation of each resource and indications of system bottlenecks. The user interface displays the results graphically. It also provides screen based facilities to manipulate directly the placement of data by redistributing fragments of data between processors as well as discs.

The queries of the benchmark are entered in the form of query trees. Each query tree represents a part of the corresponding source query. A query is expressed as a list of query trees preceded by a list of query parameters. Each node in a query tree represents a data operation, such as Join, Select or Project.

STEADY is able to provide users with testbeds for various DBMS architectures. A part of the interface is devoted to the creation of new DBMS libraries which are used by the Evaluator at a later stage. The interface allows the various parameters and/or features of the RDBMS architecture to be specified, including the hardware architecture, lock manager, file system, operating system and database kernels. These are then used to construct a program for the new architecture and compile it into a library. When invoked, the compiled architecture library is linked with other parts of the Evaluator to predict the system throughput for a given benchmark. Thus far a library for the Goldrush Ingres system has been implemented and validated.

3.1 DPtool

The data placement module, DPtool, enables users to exercise different data placement strategies so that the best strategy may be chosen for their applications. It uses information on the DBMS architecture, relations and queries, together with the data placement strategy selected by the user. From this it produces a data placement scheme. Together with other modules, it can be used to evaluate various data placement schemes. Users may vary the data placement strategy in order to obtain the strategy which leads to the most favourable system throughput.

DPtool enables a user to choose a specific combination of declustering, placement and redistribution methods and run DPtool to obtain a data placement scheme for the given set of relations and queries under the given architecture. Since there might be special requirements for some data placement strategies, some strategies may not be suitable for particular DBMS architectures or for certain kinds of queries. Therefore, the system may recommend some data placement strategies after analysing the queries and/or the DBMS architecture. This feature makes DPtool a useful decision support tool.

DPtool consists of three sub-modules: *Declustering*, *Placement* and *Re-distribution*. They are described as follows:

Declustering sub-module consists of a number of declustering methods. The choice of a declustering method causes the input relations to be partitioned into fragments which will be allocated to the processors of a parallel machine by the Placement and/or Re-distribution sub-modules. Only relation-level granularity and horizontal partitioning are currently taken into account in this sub-module. A number of declustering methods have been incorporated into the declustering sub-module. They include *user specified fragmentation*, *hash partitioning*, *range partitioning*, *hybrid-range partitioning (HRPS)* [18] and *EDBS declustering* [16].

Placement sub-module consists of a number of placement strategies, including *Tuple-based Round-Robin*, *Fragment-based Round-Robin*, *Apers* [13], *Bubba* [17], *Hua* [15] and *EDBS* [16]. Apart from these, several enhanced strategies are also supported. These are: (1) *network traffic based*: derived from Apers specifically for ICL Goldrush Megaserver; (2) *size based*: derived from Hua specifically for Goldrush; (3) *access frequency based*: derived from Bubba specifically for Goldrush; (4) *size and access frequency based*: derived from Hua and Bubba specifically for Goldrush. Among these strategies, some can only be used with particular declustering methods. For example, the EDBS strategy utilises hash-based declustering. Therefore, this sub-module restricts the choice of placement strategy to those compatible with the declustering strategy chosen.

Re-distribution sub-module gives users a choice among a number of re-distribution procedures. The re-distribution sub-module takes as input an old placement scheme and produces a new data placement scheme in such a way as to achieve a compromise between load balancing and overall load reduction under new situations. It consists of a number of re-distribution procedures, such as *Bubba* [17] and *Hua* [15]. Like the placement sub-module, it also restricts choice among re-distribution procedures to those compatible with the declustering method chosen by the user.

The data placement schemes produced by the DP-tool module are used as input to the Modeller. Users can also directly alter the data layout by modifying the data placement scheme through the graphical user interface. This provides for flexibility so that additional factors, such as the logging strategy employed, can be taken into account when placing relations across multiple processors and discs.

3.2 Profiler

The Profiler is responsible for maintaining statistical profiles of both base and temporary relations. It estimates the profiles for temporary relations arising from data operations in a query, using the profiles of the relations acted on by the data operations. These temporary relation profiles are used by the Modeller to estimate the cost of the data operations and the number of I/O operations which are performed remotely.

Fig. 2 illustrates the structure of the Profiler. Its two sub-modules are responsible for maintaining base relation and temporary relation profiles respectively.

Base relation profiler generates statistical profiles for all base relations. It can be run independently as a standalone program. The base relation profiles it produces can only be accessed and modified by the temporary relation profiler. Like the APE profiler [1], the base relation profiler initially generates base relation profiles with a 100 percent confidence level. As the database is updated, the confidence level decreases. Thus, when base relation profiles are used, the graphical user interface adopts a confidence level which a user hopes to reach. This confidence level value is used to decide if the current base relation profiles, which the base relation profiler maintains, need to be re-generated. If the supplied confidence level is higher than that of the base relation profiles, the base relation profiler will be invoked to re-generate the profiles in order to reach the required confidence level.

Temporary relation profiler is mainly responsible for generating statistical profiles of relations generated dynamically during query execution. It has been implemented as a library which is linked with the Modeller. When the temporary relation profiler is invoked to estimate a temporary relation profile resulting from a data operation, it takes as input the data operation as well as the operand relation profiles which might be base or temporary. The predicates used in the data operation are analysed in order to estimate the selectivity of the data operation. In the case of update operations, the relevant profiles are modified to reflect the effect of updates.

3.3 Modeller

The Evaluator needs to know the I/O access and data operation costs for an average transaction in the given benchmark or application before it can estimate system throughput values. The Modeller estimates these costs automatically. It provides a means to deal with new sets of benchmarks or applications as well as new data placement schemes. The generated costs are stored in a file as a workload profile for the given benchmarks or applications (hereafter the term "benchmarks" will be used to refer to "benchmarks or applications").

The costs associated with the execution of a benchmark generated by the Modeller include I/O access costs as well as data operation costs. I/O costs are characterised in terms of a number of basic costs such as logging costs, locking costs and costs for read/write accesses. Data operation costs refer to the CPU time consumed by non-I/O data operation processes (database backend).

Other costs the Modeller estimates include the average number of context switches per transaction and the initial cost of processing the transaction request.

Some benchmarks, such as TPC-C, consist of a set of queries. For these benchmarks, all the queries are required to be entered into the Modeller in order to generate the corresponding benchmark workload profile. Given a query and the data placement scheme produced by DPtool, the Modeller uses these to estimate the number of local and remote I/O accesses for the query running on each PE. It also estimates the data operation time (excluding I/O access time) for the query. This is a more complicated process. The query is first analysed and the size of each temporary relation is estimated (in other words, the selectivity of each operation is estimated). The selectivity estimation is performed by invoking the temporary relation profiler. To estimate the data operation cost, a set of cost formulae is also required for each type of operation, such as **Join**, **Select** and **Project**. These formulae are obtained from the corresponding DBMS library. The total cost of all the operations required in the query is taken as the data operation cost in the workload profile.

The benchmark workload profile generated by the Modeller can be added to a library so that a user can retrieve it directly at any time, and execute the Evaluator to obtain the estimated system throughput for this particular benchmark. However, the profile is also subject to modifications when the placement of relations is changed.

For multi-query benchmarks, the Modeller produces an estimate of the costs for an average transaction after it has estimated the cost of each query. Suppose a benchmark consists of multiple queries, then the cost of an average transaction in the benchmark is the appropriately weighted average cost determined from the costs of the individual queries. In addition, for each processing element a cost is estimated for an average transaction, reflecting the situation that the average transaction runs on the corresponding processing element.

Each query in a benchmark is represented by a list of query trees. Post-order traversal is adopted when the Modeller analyses a query tree. Each operator is divided into a number of steps and the costs for each step accumulated to produce the overall cost of the operation. For example, in estimating the cost of a **Select** operation, the Modeller divides the cost of the operation into three parts: *access cost*, *predicate evaluation cost* and *storage/transfer cost*. These three costs together form the data operation cost of the operator. Of these, the access cost consists of the costs incurred by the access methods in finding the data pages required. The storage/transfer cost refers to the cost of storing or transferring the result tuples. The predicate evaluation cost refers to the cost of evaluating the retrieved tuples against a given predicate. A conjunctive predicate is treated using the algorithm of Whang [21] which provides a probabilistic estimate on the number of terms evaluated in a conjunctive expression. The main idea is that if a term in the sequence is false then remaining predicates need not be evaluated. Assuming a probability of 0.5 that a term fails, the number of terms evaluated is:

$$E_t = \frac{2^{N_t} - 1}{2^{N_t - 1}}$$

where E_t is the number of terms evaluated, N_t is the number of terms in the predicate.

I/O costs are not included in the above three costs. They take the form of the number of reads/writes and lock requests/replies the operator requires. The estimation of I/O costs is based on the number of pages which the operator may need to read or write. This in turn depends on the estimation of selectivity or result size, as well as some attribute values of the result tuples.

Estimating the number of local and remote I/O requests requires knowledge of selectivity from relation fragments, given a **Select** predicate. Furthermore, when a **Select** predicate involves user input variables (e.g. in TPC-B, all accesses to base relations are based on the account_id, branch_id, teller_id given by users), it is difficult to estimate accurately the number of local and remote I/O requests with respect to a particular PE simply based on data placement information. The base and

temporary relation profiles maintained by the Profiler provide information for this task. The Modeller can obtain the selectivity estimates for a particular relation fragment through the histograms of the relation (in the case of non-parametric profiling) or the attribute distribution type (in the case of parametric profiling). The value distribution of the user input variable can also be obtained from the distribution of the corresponding relation attribute, if there is a relation attribute related to the input variable.

3.4 Evaluator

The Evaluator estimates system throughput for a given workload and a particular DBMS architecture. It does this by estimating the utilisation of each resource and identifying the bottlenecks in the system. The metric of system throughput is tps (transactions per second). The capability to identify bottlenecks is particularly useful for exercising and evaluating different data placement strategies, since different data placement schemes may lead to the same cumulative consumption of resources, but different consumption of an individual resource. Good data placement will lead to an even distribution of consumption over all resources and therefore reduce the possibility of poor performance due to uneven loading.

Fig. 3 shows the structure of this module. It consists of 3 parts, *Set-Up module*, *DBMS architecture library* and *Miscellaneous values model*, with *benchmark workload profiles* taken as input.

The Set-Up module combines the selected DBMS architecture library with the given workload profile to produce the estimated throughput value on the DBMS architecture. Upon receiving a request to process a workload profile with a DBMS architecture, the Set-Up module computes an average transaction of that benchmark on each PE in turn and records the consumption of each resource in the transaction running on each PE. From these results, the bottlenecks in the system are identified and the throughput estimate is based on the usage of these bottleneck resources.

A DBMS architecture library is used to calculate the usage of resources within the corresponding DBMS architecture, given a workload profile. This library can be invoked by the graphical user interface during run time. The DBMS architecture library is based on three performance models: the *file system performance model*, the *lock manager performance model* and the *target primitive values model*. The file system performance model contains a set of formulae concerned with the times for file system calls, such as local reads, remote reads, local writes and remote writes. This also includes the formulae for log reads and log writes. The lock manager performance model contains formulae for calculating the times for lock requests and replies. The target primitive values model contains a set of target primitive values for a parallel DBMS. These might include processing unit costs associated with interaction with the communication control unit, communication control unit costs associated with network driving, communication transmission costs, and so on.

4 Applications of STEADY

STEADY is designed as a tool for DBMS design as well as database administration. It is also useful in capacity management of parallel database systems in that it can be used to tune the system to achieve better performance and to warn of impending problems before they occur. The following sub-sections describe a few applications of STEADY.

4.1 Estimate throughput from a set of queries

To estimate system throughput for a benchmark or a set of queries over a database, users must input the DBMS architecture, details of the structures of the database, the queries to be used and choice of data placement strategy through the graphical user interface. The Profiler is then used to produce base relation profiles for the database while DPtool produces a data placement scheme for all relations involved. From this, the Modeller produces a workload profile. Finally, the Evaluator estimates utilisation of each resource and hence the system throughput.

4.2 Estimate throughput from an existing benchmark workload profile

In cases where a workload profile for a benchmark with a particular DBMS architecture and a particular data placement scheme has already been produced, a user only needs to run the Evaluator to estimate the system throughput. This is particularly useful for some simple benchmarks where users can estimate various costs by themselves. Moreover, when a benchmark (or set of queries) cannot be handled by the Modeller (e.g. because of use of some special operators), a user can obtain the relevant workload through experiments using available systems and then directly run the Evaluator on the actual workload profile to get the estimated throughput values for the target system.

4.3 Manipulate data layout

Users can exercise different data placement strategies for the same benchmark by running DPtool, the Profiler, the Modeller and the Evaluator in turn to estimate system throughput for each data placement strategy. The strategy with the greatest throughput can be chosen for running the benchmark on the actual platform. Since some data placement strategies may only be suitable for a particular DBMS architecture, the choice of unsuitable data placement strategies will be disabled and only data placement strategies which are known to be suited to the given DBMS architecture will be used.

Users can also manipulate the data layout directly through the graphical user interface by moving relation fragments across multiple processors and discs. This provides a fine tuning facility for data placement after a data placement strategy has been chosen to produce an initial data layout. It is useful as some additional factors, such as the logging strategies employed by the DBMS, are currently not taken into account by DPtool when using the chosen data placement strategy to produce the initial data layout.

4.4 Tune system parameters

STEADY can also be used to tune some system parameters for DBMSs under development. This can be achieved by manipulating the values of some particular parameters in a DBMS library and comparing the system throughput under different values. The graphical user interface provides a convenient means for users to alter the values of the system parameters. It displays detailed information about system performance (e.g. throughput, utilisation of each resource) under different values of a particular system parameter in histogram form.

5 Use of STEADY in Data Placement Study

This section presents some results of a study of several types of data placement strategies to demonstrate the usefulness of STEADY in comparative studies on parallel database systems. In this study, STEADY was configured to support a shared-nothing DBMS architecture utilising inter-transaction parallelism and used to predict performance for various data layouts obtained using three different types of data placement strategies. The number of PEs participating in database activities under such a DBMS architecture was varied from 1 to 60 in order to investigate the relative scalability of different data placement approaches. The focus of this study of data placement strategies is on update intensive transaction processing applications, as simulated by TPC-B and TPC-C.

The data placement strategies chosen for the study represent three types: *size based* (hua), *access frequency based* (heat) and *network traffic based* (apers). The number of fragments of the relations is chosen as 70, which is larger than any number of PEs considered.

For TPC-B a data set was generated consisting of four relations. The size of the largest relation (i.e. *Account*) is 2000K bytes. Using each of the data placement strategies in turn the number of PEs was varied from 1 to 60 and the system throughput determined for each case. The results obtained are shown graphically in Fig. 4. A similar exercise

was conducted for the TPCC benchmark, and the results are displayed in Fig. 5. The bottlenecks predicted by STEADY are discs attached to particular PEs in each case.

For TPC-B, the performance obtained from the *network traffic based* strategy varies in a similar way to that obtained from the *size based* strategy with minima when the number of PEs is equal to 6, 8, 12, 15, 18, 24 and 36, and maxima at 5, 7, 10, 14, 17, 23 and 35 PEs. There are slight differences between the performance predictions obtained using *network traffic based* and *size based* strategies for numbers of PEs below 35, and more noticable differences for numbers above 35. The maximum difference in performance obtained using these two types of strategies is about 22%. However, an average difference in performance is 1.6%.

The *access frequency based* strategy is not subject to performance minima which the *size based* and *network traffic based* strategies give rise to for TPC-B. The performance obtained using the *access frequency based* strategy is generally better than that obtained using the other two types of strategies.

For TPC-C, when the number of PEs is less than 35, all three types of strategies produce similar performance behaviour, with the *access frequency based* strategy producing the worst performance in almost all cases. For numbers of PEs above 35, there is significant variation (maximum variation is 36.6%) between performances provided by these three approaches.

The performance curve for TPC-C provided by the *network traffic based* strategy shows a delay in scaling up and a few noticeable drops, compared with that provided by the *size based* strategy. The *access frequency based* strategy provides relatively poor performance in the case of TPC-C.

From the results presented here it is clear that (a) the choice of data placement strategy can have a significant effect on the overall performance of the database system; (b) the choice of strategy will vary with different applications. In this case the *access frequency based* strategy produced the best performance for TPC-B and the worst for TPC-C. A more detailed account of the results of this study which covers other data placement strategies and provides a detailed analysis of the performance results will be given in a separate paper [22].

6 Summary and Concluding Remarks

STEADY is a convenient and flexible tool for performance estimation of shared-nothing parallel database systems. It consists of four major modules: DPtool, the Profiler, the Modeller and the Evaluator. DPtool can be used to generate various data placement schemes using different strategies. The Profiler is primarily responsible for estimating the number of tuples resulting from data operations. It assists the work of the Modeller, which is responsible for producing the benchmark workload profile for a particular benchmark or application. The Evaluator takes in workload profiles and predicts the utilisation of each resource and from it the system throughput values. Currently, a prototype of STEADY has been implemented, which supports the ICL Goldrush Megaserver running INGRES.

There are several limitations of the analytical model adopted in the prototype of STEADY, which need further investigation. Firstly, in the current analytical model, the cache hit ratio is assumed to be constant and context independent. However, slight variations in it are to be expected and might have a significant impact on performance. Secondly, there is no attempt to model contention for locks. Transaction failure and rollback to avoid deadlock are also not modelled. Finally, it is assumed that at most one factor is a bottleneck during a transaction. However, a single transaction could be bound by different bottlenecks at different stages. For example, a transaction could be disc bound, then CPU bound, then disc bound and then network bound during its execution process. It is planned to conduct further investigation into the effects of the above mentioned factors and improve the analytical model by taking account of these factors in modelling relevant stages of the execution of a transaction.

Work is being carried out to enhance the func-

tionality of STEADY. This includes designing and implementing a histogram-based relation profiler to replace the parametric-distribution-based profiler currently in use, and developing a user-friendly graphical user interface. It is planned to add a paralleliser component to the system in order to support some DBMSs such as the EDBS [12]. STEADY is also being integrated with an automated search driving tool called Testpilot [23] to allow studies to be made of how sensitive throughput estimations are to variations in the values of configuration parameters.

7 Acknowledgements

The authors acknowledge the support received from the European Community under the Esprit III programme for the Pythagoras project and from the Engineering and Physical Sciences Research Council under the PST programme. Particular thanks are due to Mr P. Broughton, Mr P. Watson and Mr A. Fitzjohn of ICL and Professor M. Kersten of CWI for their support and assistance with the work.

8 References

[1] F. Andres. M. Couprie, and Y. Viemont. A multi-environment cost evaluator for parallel database systems. in A. Makinouchi, editor, *Database Systems for Advanced Applications'91*, World Scientific, Singapore, pages 126–135, 1991.

[2] B. Bergsten, M. Couprie, and P. Valduriez. Prototyping DBS3, a shared-memory parallel database system. *Proceedings of the 1st Int. Conf. on Parallel and Distributed Information Systems*, Florida, December 1991.

[3] K. F. Wong and M. Paci. Performance evaluation of an OLTP application on the EDS database server using a behavioural simulation model. *ECRC Technical Report*, 1991.

[4] Transaction Processing Performance Council (TPC). TPC benchmark B. In J. Gray, editor, *The Benchmark Handbook for Database and Transaction Processing Systems (2e)*. Morgan Kaufmann, San Mateo, CA, 1993.

[5] Transaction Processing Performance Council (TPC). TPC benchmark C. In J. Gray, editor, *The Benchmark Handbook for Database and Transaction Processing Systems (2e)*. Morgan Kaufmann, San Mateo, CA, 1993.

[6] M. Stonebraker. The case for shared nothing. *Database Engineering Bulletin*, 9(1):4–9, March 1986.

[7] S. Dietrich, M. Brown, E. Cortes-Rello, and S. Wunderlin. A practitioner's introduction to database performance benchmarks and measurements. *The Computer Journal*, 35(4):322–331, August 1992.

[8] C. Turbyfill, C. Orji, and D. Bitton. AS3AP: an ANSI SQL standard scaleable and portable benchmark for relational database systems. In J. Gray, editor, *The Benchmark Handbook for Database and Transaction Processing Systems (2e)*. Morgan Kaufmann, San Mateo, CA, 1993.

[9] D. Sacca and G. Wiederhold. Database partitioning in a cluster of processors. *Proceedings of the 9th VLDB Conference*, pages 242–247, Florence, Italy, 1983.

[10] H. Boral, W. Alexander, L. Clay, G. Copeland, S. Danforth, M. Franklin, B. Hart, M. Smith, and P. Valduriez. Prototyping bubba, a highly parallel database system. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):4–24, March 1990.

[11] D. DeWitt, S. Ghandeharizadeh, D. Schneider, A. Bricker, H. Hsiao, and R. Rasmussen. The Gamma database machine project. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):44–62, March 1990.

[12] F. Andres, B. Bergsten, P. Borla-Salamet, P. Broughton, C. Chachaty, M. Couprie, B. Finance, G. Gardarin, K. Glynn, B. Hart, S. Kellett, S. Leunig, M. Lopez, M. Ward, P. Valduriez, and M. Ziane. EDS — collaborating for a high performance parallel relational database. *Proceedings of the Annual ESPRIT Conference*, pages 274–295, Brussels, November 1990.

[13] P. Apers. Data allocation in distributed database systems. *ACM Transactions on Database Systems*, 13(3):263–304, September 1988.

[14] K. Hua and C. Lee. An adaptive data placement scheme for parallel database computer systems. *Proceedings of the 16th VLDB Conference*, pages 493–506, Brisbane, Australia, 1990.

[15] K. Hua, C. Lee, and H. Young. An efficient load balancing strategy for shared-nothing database systems. *Proceedings of Database and Expert Systems Applications 92*, pages 469–474, Valencia, Spain, 1992, Springer-Verlag.

[16] M. B. Ibiza-Espiga and M. H. Williams. Data placement strategy for a parallel database system. *Proceedings of Database and Expert Systems Applications 92*, pages 48–54, Valencia, Spain, 1992, Springer-Verlag.

[17] G. Copeland, W. Alexander, E. Boughter, and T. Keller. Data placement in Bubba. *SIGMOD Rec.*, 17(3):99–108, September 1988.

[18] S. Ghandeharizadeh and D. DeWitt. Hybrid-range partitioning strategy: A new declustering strategy for multiprocessor database machines. *Proceedings of the 16th VLDB Conference*, pages 481–492, Brisbane, Australia, 1990.

[19] P. Mishra and M. Eich. Join processing in relational databases. *ACM Computing Surveys*, 24(1):63–113, March 1992.

[20] M. Mannino, P. Chu, and T. Sager. Statistical profile estimation in database systems. *ACM Computing Surveys*, 20(3):191–221, September 1988.

[21] K-Y Whang, R. Krishnamorthy. Query optimization in a memory-resident domain relational calculus database systems. *ACM TODS*, 15(1):67–75, March 1990.

[22] S. Zhou, M. H. Williams, and H. Taylor. A comparative study of data placement in a parallel DBMS. *submitted for publication to IEEE Transactions on Computers*, 1995.

[23] M. Kersten and F. Kwakkel. Design and implementation of a DBMS performance assessment tool. *Proceedings DEXA '93*, pages 265–276, Prague, September 1993.

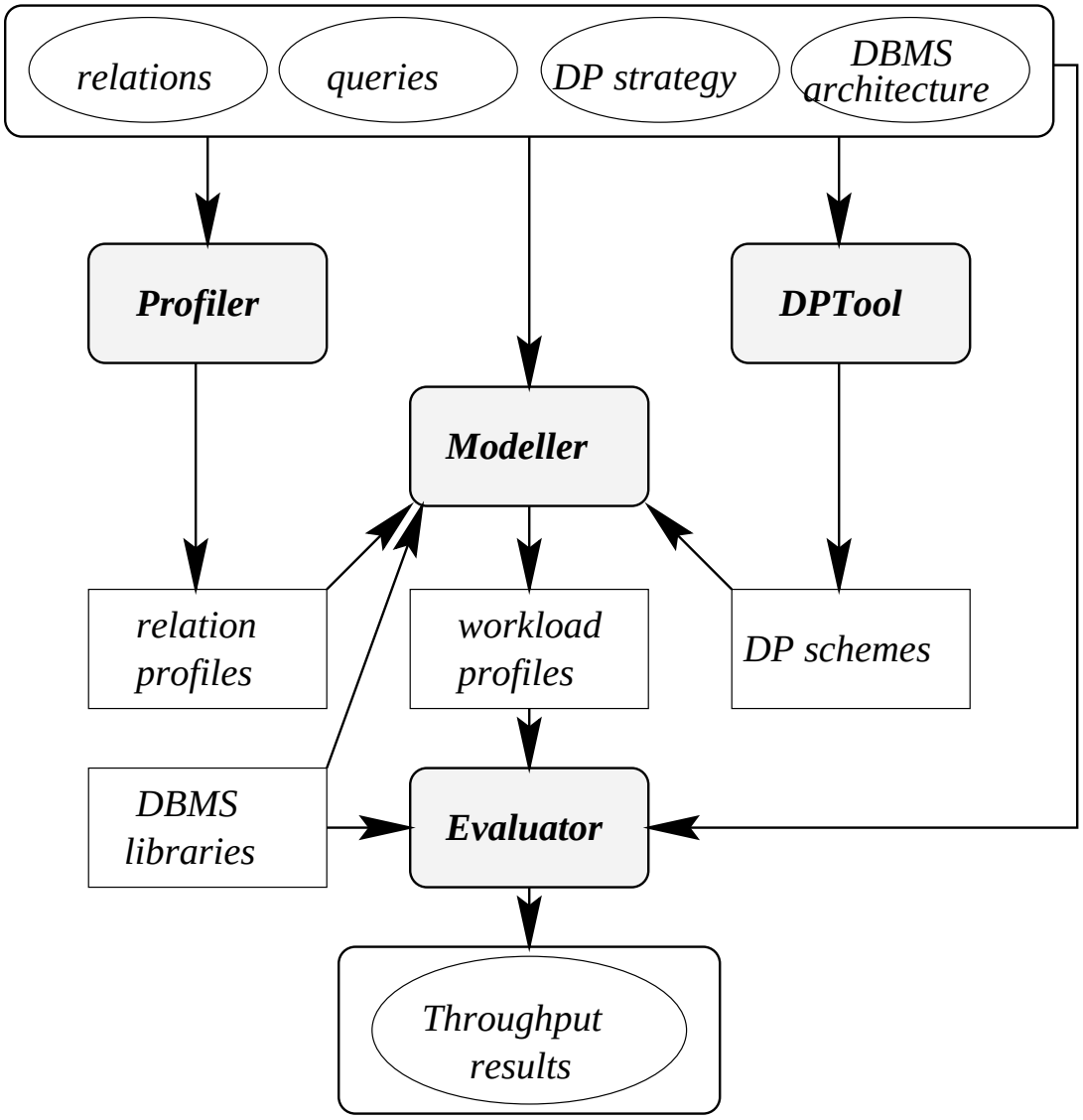

Figure 1. Architecture of Software STEADY

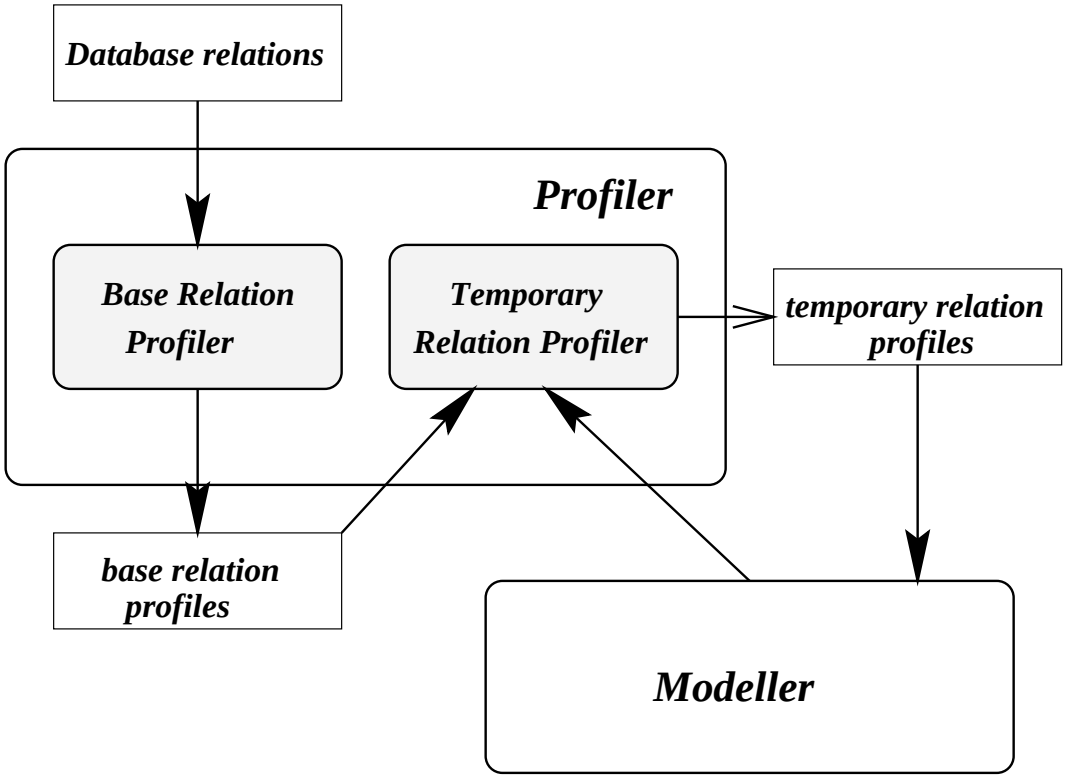

Figure 2. Structure of the Profiler

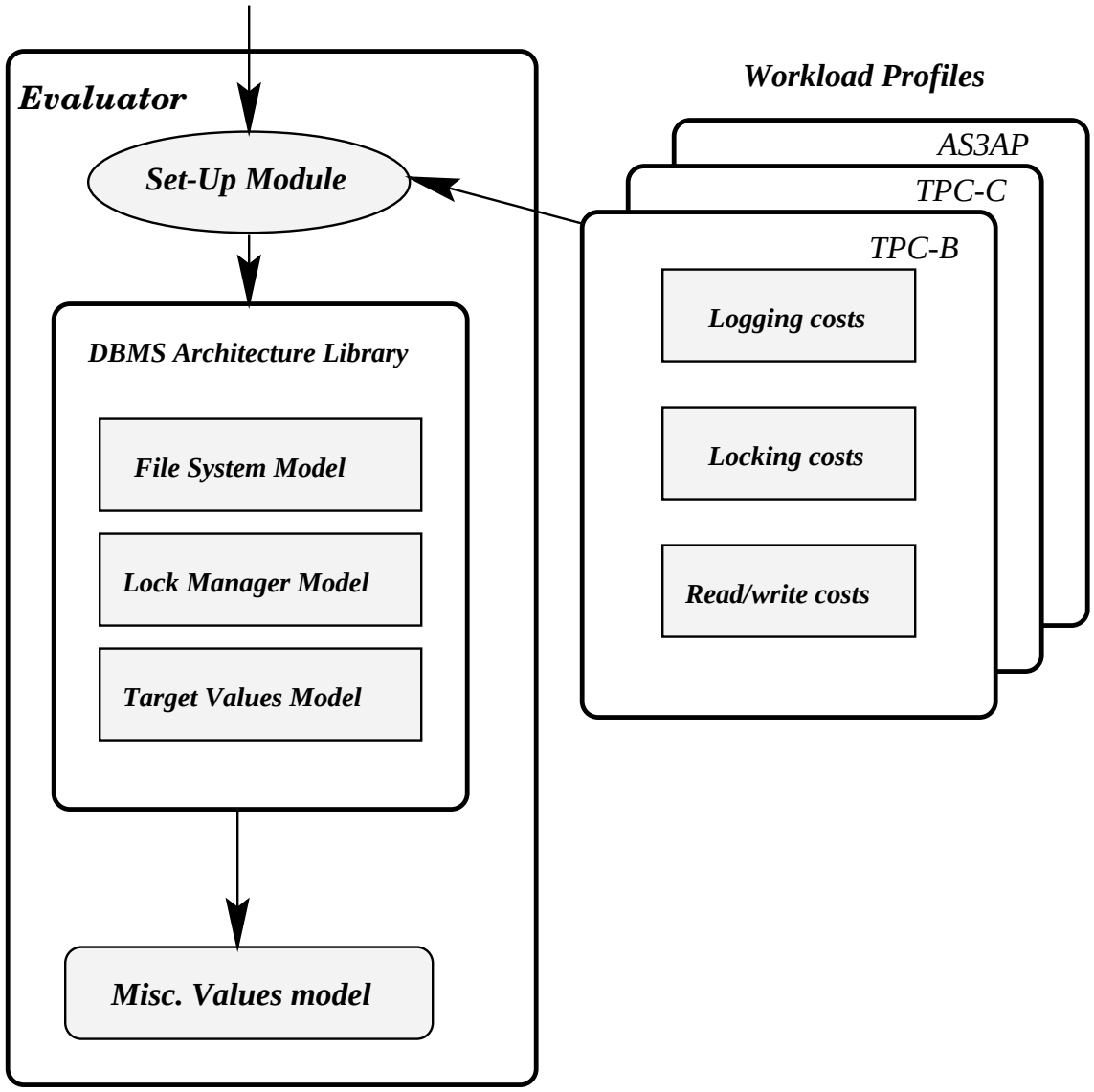

Figure 3. Structure of Evaluator

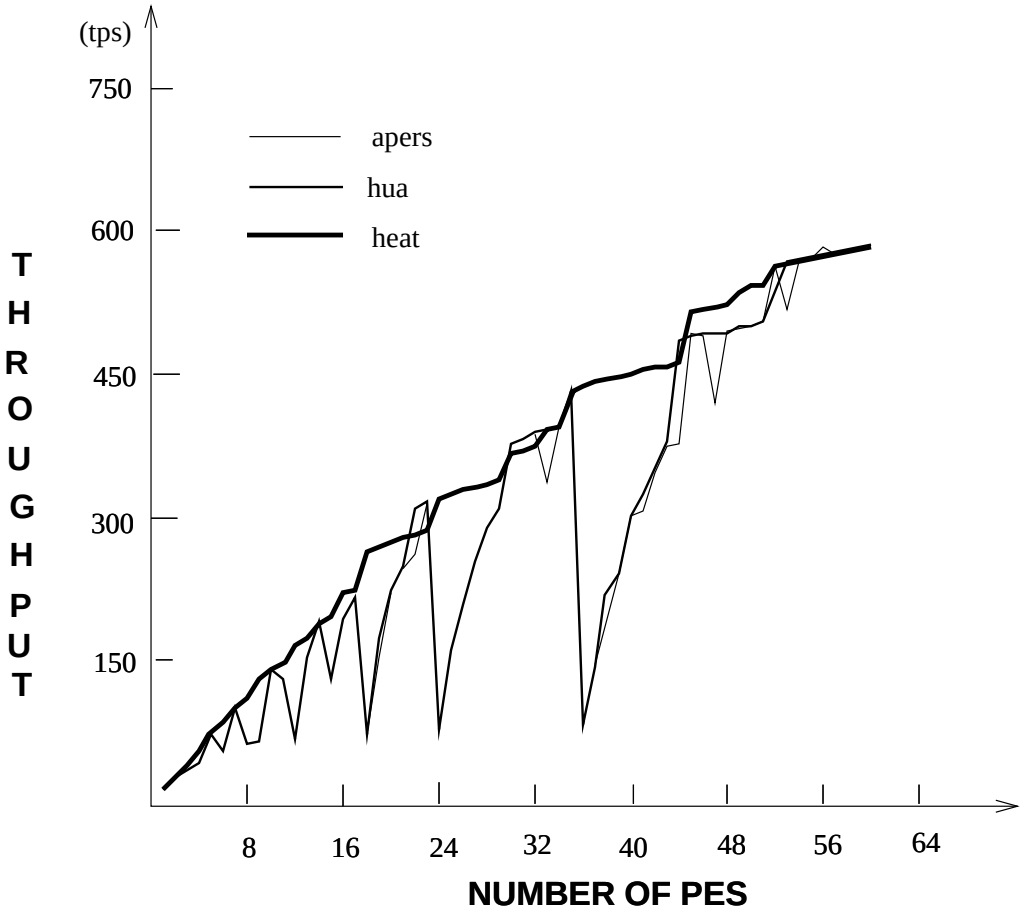

Figure 4. Performance of Strategies of Different Types for TPC-B

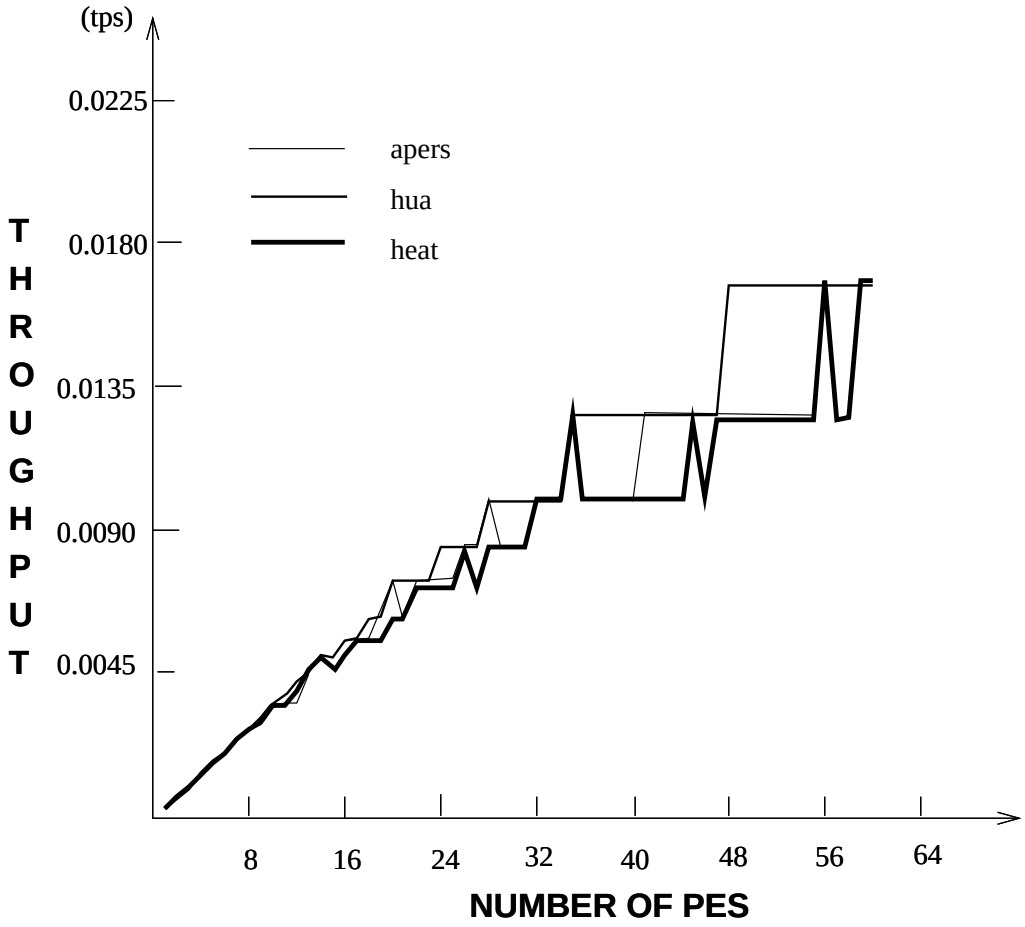

Figure 5. Performance of Strategies of Different Types for TPC-C